Software–Projekt 2005/06

VAK 03-901.01

Testplan

GLOBUS 4

Kamal Badawi	kbadawi@tzi.de	1812949
Teodosiy Kirilov	tdik@tzi.de	1967429
Tobias Lindner	tlindner@tzi.de	1871150
Viktor Schwam	viktors@tzi.de	1942734

Abgabe: 11. Mai, 2006

Inhaltsverzeichnis

0 Versions- und Änderungsgeschichte

Datum	Extern	Revision	Beschreibung
11.05.2006	1.0	NV	Erste Veröffentlichung des Plans

1 Einführung

1.1 Zweck

Der Zweck dieses Dokumentes ist es, eine genaue Definition der zu absolvierenden Tests der Software zu erstellen. Des Weiteren ist dieser Testplan dazu gedacht, genau festzulegen, was genau an der zu entwickelnden Software getestet werden soll und wie umfangreich diese Tests ausfallen sollen, so dass kein Missverständnisse mit dem Kunden auftreten können. Dieses Dokument ist sowohl für den Kunde als auch für alle an der Implementierung der Software beteiligten Mitarbeiter unserer Firma gedacht. Diese betrifft sowohl die Entwickler, die Programmierer, die Rechtsabteilung, die Designer und die übrigen Mitarbeiter..

1.2 Referenzen

Aufbau des Testplans:
http://www.informatik.uni-bremen.de/st/Lehre/swp/abgabe4.html

Definitionen:
http://www.wikipedia.de

IEEE Standard:
IEEE Standard 830-1998: IEEE Standard for Software Test Documentation

Inhaltliche Struktur:
http://www.informatik.uni-bremen.de/st/Lehre/swp/abgabe4.html
Vorlesung Software-Projekt Universität Bremen 05/06

2 Verbindung zu anderen Dokumenten

2.1 Anforderungsspezifikation

Der Testplan basiert auf der zuvor erstellten Anforderungsspezifikation. Es soll durch diesen Testplan sichergestellt werden, dass sämtlich Funktionalität und Funktionsumfang so gewährleistet werden kann, wie es bereits in der Anforderungsspezifikation mit dem Kunden vereinbart wurde. Um dieses zu gewährleisten, wird genau festgelegt, wo die kritischen Bereiche des zu entwickelnden Systems liegen und in welcher Art und Weise diese durch Tests überprüft werden, um eine reibungslose Funktionalität zu garantieren. Es wird genau festgelegt welchen Umfang die vorgesehenen Tests haben müssen, denn nur so kann sichergestellt werden, dass es sich bei der Übergabe an den Kunden um ein stabiles und fehlerfreies Softwaresystem handelt.

2.2 Architekturbeschreibung

Dieser Testplan basiert ebenfalls auf der bereits abgegebenen Architekturbeschreibung. Dort wurden die Komponenten, welche jetzt getestet werden sollen festgelegt. In der Architekturbeschreibung wurden sämtliche systemkritischen Komponenten detailliert beschreiben, so dass der Testplan auf dieser Beschreibung aufbaut. Es werden die gleichen Bezeichnungen wie in der Architekturbeschreibung verwendet und für jede Komponente genau beschrieben wie und in welchem Umfang Tests durchgeführt werden.

2.3 Namenskonventionen

JVM = Java Virtual Machine

JUnit = Framework zum Testen von Java-Programmen

GUI = Graphical User Interface

Black-Box-Test = Eine Methode des Software-Tests

White-Box-Test = Eine weitere Methode des Software-Tests

3 Systemüberblick

Hier werden einige Komponenten unseres Systems aufgelistet, die durch Komponententest getestet werden sollen:

1. Administration
   Es sollen Komponenten getestet werden, die die Administration des Systems beinhalten. Das sind folgende:

   1.1 User_administration
   Diese Komponente dient zur Verwaltung von Benutzern.

   1.2 Reference_administration
   Diese Komponente dient zur Verwaltung von Referenzen.

   1.3 DB_administration
   Diese Komponente dient zur Sicherung und Wiederherstellung der Datenbank.

   1.4 Sanity_Check
   Diese Komponente dient zur Redundanz- und Inkosestenzprüfung.

2. UserManagement
   Diese Komponente dient dem Verwalten von bereits registrierten Benutzern auf dem System.

3. ArticleManagement
   Diese Komponente sorgt für die Verwaltung von Artikeln in der Datenbank.

4. SearchEngine
   Diese Komponente soll die Suchanfragen der Benutzer verarbeiten und Ergebnisse liefern.

5. GUI
   Dies ist die Graphische Benutzeroberfläche und dient dem einfachen navigieren durch das System.

4 Testmerkmale

4.1 Zu testende Merkmale

Hier werden einige Merkmale und Kombinationen von Merkmalen unseres Systems aufgelistet, die getestet werden sollen:

1. Administration
   - 1.1 Benutzerverwaltung
     - 1.1.1 Einen Benutzer löschen.
     - 1.1.2 Einen Benutzer sperren.
       - 1.1.2.1 Einen gesperrten Benutzer löschen.
       - 1.1.2.2 Einen gesperrten Benutzer wieder freigeben.
       - 1.1.2.3 Profil eines gesperrten Benutzers ändern.
     - 1.1.3 Profil des Benutzers ändern.
   - 1.2 Referenzverwaltung
     - 1.2.1 Neue Referenz einfügen.
     - 1.2.2 Referenz bearbeiten.
     - 1.2.3 Referenz löschen.
   - 1.3 Datenbank
     - 1.3.1 Datenbankinhalt sichern.
     - 1.3.2 Datenbankinhalt wiederherstellen.
   - 1.4 Redundanz- und Inkosistenzprüfung
     - 1.4.1 Alle Artikel in der Datenbank überprüfen.
     - 1.4.2 Mehrfach vorhandene Artikel entfernen.
     - 1.4.3 Fehlerhafte Artikel korrigieren.
2. UserManagement
   - 2.1 Neuen Benutzer im System registrieren.
   - 2.2 Benutzerprofil bearbeiten.
   - 2.3 Benutzer komplett aus dem System löschen.
   - 2.4 Bei vergessenen Passwörtern sich ein Neues zuschicken lassen.
     - 2.4.1 Anmelden am System mit dem alten Passwort nach Zusendung eines neuen Passworts.

2.5 Benutzer im System anmelden.

2.6 Benutzer im System abmelden.

3. ArticleManagement

3.1 Einen neuen Artikel in die Datenbank einfügen.

3.2 Einen Artikel in der Datenbank bearbeiten.

3.3 Ein Artikel im BibTex Format in die Datenbank importieren.

3.4 Einen Artikel in einer geeigneten Form anzeigen lassen.

3.5 Taxonomy

3.5.1 Eine Liste mit allen Referenzen aus der Datenbank anzeigen lassen.

3.5.2 Exportieren der Taxonomy.

3.5.3 Bearbeiten der Taxonomy.

3.6 Article

3.6.1 Kommentare eines Artikels anzeigen lassen.

3.6.2 Kommentar zu einem Artikel verfassen.

3.6.3 Kommentar eines Artikels bearbeiten.

3.6.4 Kommentar zu einem Artikel löschen.

3.6.5 Einen Artikel mit allen seinen Kommentaren löschen.

3.6.6 Exportieren des Artikels im BibTex Format.

3.6.7 Einen Artikel bewerten.

4. SearchEngine

4.1 Standartsuche nach einem Artikel.

4.2 Standartsuche mit Wildcards nach einem Artikel.

4.3 Erweiterte Suche nach einem Artikel.

4.4 Erweiterte Suche mit Wildcards nach einem Artikel.

4.5 Finden aller Artikel eines Autors.

4.6 Finden aller eingetragenen Artikel eines Benutzers.

4.7 Finden aller Artikel einer Kategorie

4.2 Nicht zu testende Merkmale

Hier werden einige Merkmale und Kombinationen von Merkmalen unseres Systems aufgelistet, die nicht getestet werden müssen:

1. Administration
   - 1.1 Benutzerverwaltung
     - 1.1.1 Profil eines nicht vorhandenen Benutzers ändern.
       Es wird in diesem Fall automatisch vom System festgestellt, dass der Benutzer nicht in der Datenbank vorhanden ist und sein Profil kann gar nicht erst angezeigt werden.
     - 1.1.2 Einen nicht vorhandenen Benutzer löschen.
       Es wird in diesem Fall automatisch vom System festgestellt, dass der Benutzer nicht in der Datenbank vorhanden ist und es wird gar nicht erst die Möglichkeit gegeben diesen Benutzer zu löschen.
   - 1.2 Referenzverwaltung
     - 1.2.1 Eine nicht vorhandene Referenz bearbeiten.
       Es wird in diesem Fall automatisch vom System festgestellt, dass die Referenz nicht in der Datenbank vorhanden ist und es wird gar nicht erst die Möglichkeit gegeben diese Referenz zu bearbeiten.
     - 1.2.2 Eine nicht vorhandene Referenz löschen.
       Es wird in diesem Fall automatisch vom System festgestellt, dass die Referenz nicht in der Datenbank vorhanden ist und es wird gar nicht erst die Möglichkeit gegeben diese Referenz zu löschen.
2. UserManagement
   - 2.1 Bei vergessenen Passwörtern sich ein Neues zuschicken lassen.
     - 2.1.1 Anmelden am System mit dem alten Passwort nach Zusendung eines neuen Passworts.
       Das alte Passwort wird in der Datenbank sofort durch das neue ersetzt und ist somit automatisch nicht mehr gültig.
   - 2.2 Nicht vorhandenen Benutzer im System anmelden.
     Das System merkt automatisch, dass es keinen Eintrag für den nicht vorhandenen Benutzer in der Datenbank gibt und wird den Benutzer nicht anmelden.

5 Abnahmekriterien

5.1 Testabdeckung

Bei der Feststellung der Testabdeckung wird die Testabdeckung in C0 bis C4 Bereiche eingeteilt. Jeder dieser Bereiche stellt eine andere Art zu testen da. Ziel ist es eine möglichst hohe Testabdeckung zu erreichen, da nur so sichergestellt werden kann, dass ein stabiles System ausgeliefert wird. Je höher die jeweilige Abdeckung ist, desto wahrscheinlicher wird es, dass ein Codefehler gefunden wird. Da es zum momentanen Zeitpunkt sehr schwer einzuschätzen ist, wie viele Fehler auftreten und beseitigt werden müssen. Ziel ist es natürlich eine hohe Testabdeckung zu erreichen, um ein fehlerfreies System auszuliefern. Die Testabdeckung sollte also so hoch liegen, dass so gut wie alle Fehler im Code gefunden werden. Um ein stabiles System auszuliefern, verwenden wir natürlich unterschiedliche Arten von Testabdeckungen. In den systemkritischen Bereichen sollte die Testabdeckung nahezu vollständig sein, da jeder auftretende Fehler hier zu einem Absturz des Systems führen kann. In anderen Bereichen die keinen so großen Einfluss auf die Stabilität des Systems haben und es sich bei Fehlern in diesem Bereich um kleine Schönheitsfehler handelt. In diesen Bereichen wird die Testabdeckung natürlich deutlich geringer sein, als in den systemkritischen Bereichen.

5.2 Verhältnis Fehler/Lines-Of-Code

Das Verhältnis von Fehler zu Lines-Of-Code kann sehr stark variieren. So können in einigen Klassen deutlich mehr Fehler zu Lines-Of-Code auftreten, in anderen Klassen treten keine Fehler auf. Da wir somit die Anzahl der Fehler zu Lines-Of-Code nicht für jede Klasse einzeln festlegen, denn das würde erfordern, dass wir für jede Klasse die entwickelt werden muss ein feste Anzahl von Fehlern zu Lines-Of-Code angeben müssten. Dieser Aufwand würde den Nutzen des Ganzen übersteigen, denn es ist sehr schwer festzulegen welche Klassen davon betroffen werden. In systemkritischen Klassen ist die Anzahl der Fehler zu Lines-Of-Code natürlich deutlich geringer als in den übrigen Klassen. Insgesamt gehen wir von einem Verhältnis von 40 Fehlern auf 1000 Zeilen Code aus. Dieses Verhältnis bezieht sich auf den gesamten Code. Es kann durchaus vorkommen, dass sich diese Fehler nicht gleichmäßig über den Code verteilen, sondern das zum Beispiel in den ersten 50 Prozent des Codes 80 Prozent der Gesamtfehler stecken. Unser Ziel ist es natürlich das Verhältnis von Fehlern zu Lines-Of-Code möglichst gering zu halten und nach Möglichkeit deutlich unter den 40 Fehler pro 1000 Zeilen Code zu liegen, um eine stabile und fehlerfreie Funktionsweise der Software zu garantieren.

5.3 Budget

Das Budget für die Testphase ist auf 96000 EUR begrenzt. Dieses Buget ist für die komplette Durchführung der Testphase eingeplant und muss hierfür auch reichen. Das Buget ist so kalkuliert, dass eine Testphase mit ausreichend vielen Tests durchgeführt werden kann, so dass die Software bei Auslieferung stabil lauffähig ist und sämtliche kritischen Fehler bereinigt wurden. Die Testphase nimmt den größten Teil unseres Budget ein, da für das Testen viel Zeit und Geld draufgeht. In der Testphase bleibt jedoch abzuwiegen welche Tests sinnvoll oder welche Tests nur kleine Schönheitskorrekturen sind und somit den Rahmen unseres Budget sprengen können. Sicherlich wird auch die Anzahl der gefundenen Fehler einen Einfluss auf die Ausnutzung des Budget haben, wichtig ist hierbei jedoch dass der Budgetrahmen auf keinen Fall überzogen werden darf. Hierzu werden die Tests nach Prioritäten sortiert und auch so abgearbeitet. Kritische Fehler müssen auf jeden fall beseitigt werden, aber kleiner Schönheitsfehler können unberücksichtig bleiben, wenn der Aufwand sie zu beheben den Nutzen und vor allem die Kosten übersteigt.

6 Vorgehensweise

6.1 Black-Box-Tests

Die Black-Box-Tests werden schon recht früh in der Implementierung erstellt. Die Black-Box-Tests, welche bereits schon vor der eigentlichen Implementierung auf Basis der Spezifikation entwickelt werden können. Nachdem die Tests erstellt wurden und der implementierte Code vorliegt, kann ein bereits erstellter Black-Box-Test ausgeführt werden. Durch diesen Test kann dann bereits festgestellt werden, ob der bereits implementierte Code schon fehlerfrei läuft. Sollte dieses nicht der Fall sein, so kann der Code weiterentwickelt werden und zu einem späteren Zeitpunkt wieder durch den gleichen Black-Box-Test überprüft werden. Eine Anpassung des Black-Box-Tests ist nicht nötig, da dieser auf der Spezifikation basiert und somit unabhängig von der eigentlichen Implementierungsart ist. Der Black-Box-Test wird somit mehrmals während der Implementierungsphase auf den dafür vorgesehenen Code angewendet. Er dient dazu Abweichung von der Spezifikation. In der folgenden Grafik ist der Verlauf eines solchen Black-Box-Tests einmal dargestellt.

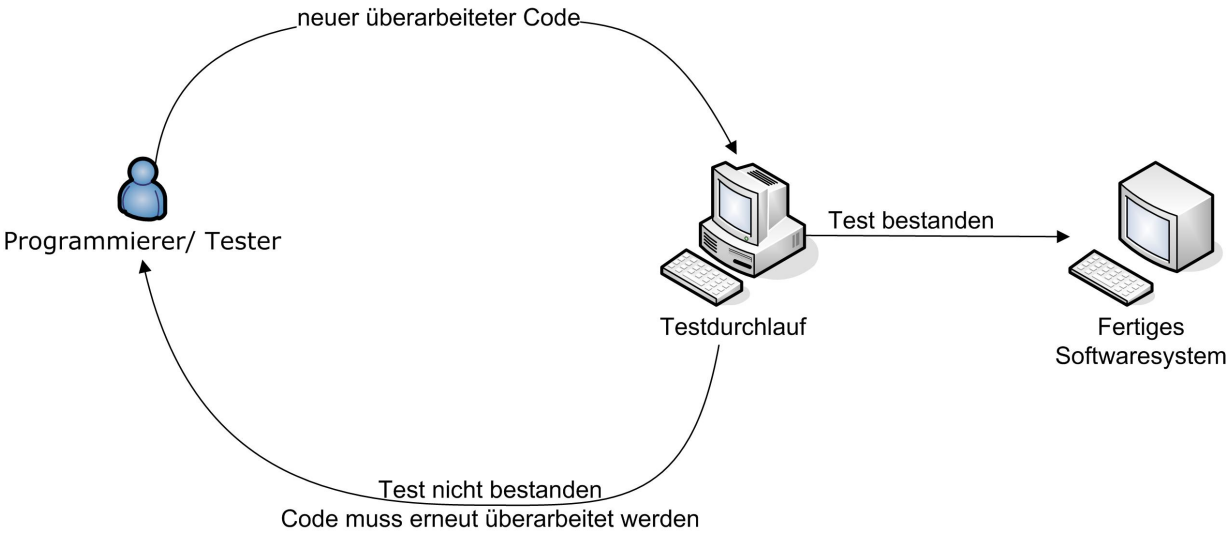

Abbildung 1: Testdurchlauf eines Black-Box-Tests

6.2 White-Box-Tests

Die White-Box-Test werden im Gegensatz zu den Black-Box-Tests noch nicht so frühzeitig entwickelt. Sie basieren auf dem implementierten Code und können somit

erst nach der eigentlichen Implementierungsarbeit entwickelt werden. Sie dienen vorallem dazu bestimmte Modul zu testen. Nach jedem Test, der einen Fehler findet und somit eine Anpassung oder Änderung am Code erforderliche macht, muss ebenfalls der White-Box-Test geändert beziehungsweise angepasst werden. Dieses ist natürlich mit einem deutlich höheren Aufwand verbunden, daher kommen White-Box-Tests nicht so häufig wie die Black-Box-Tests zum Einsatz. In der folgenden Grafik ist der Verlauf eines White-Box-Tests einmal verdeutlicht.

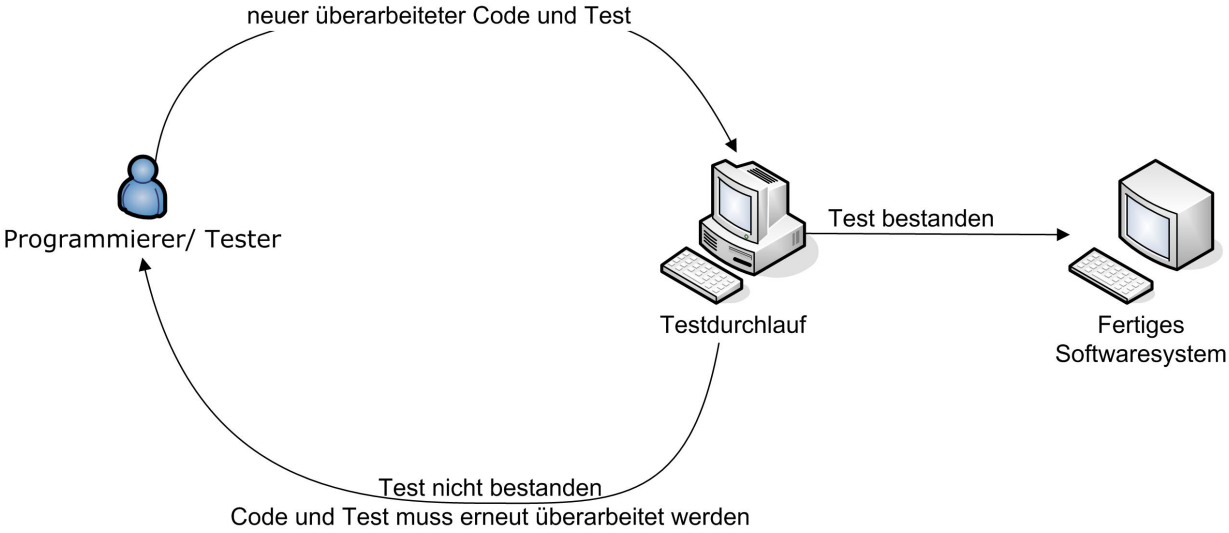

Abbildung 2: Testdurchlauf eines White-Box-Tests

6.3 Integrationsstrategien

Bei der Integrationsstrategie für die Testphase haben wir uns für die Bottom-Up Strategie entschieden. Diese Strategie wird es uns erlauben schnelle und einfach zu interpretierende Tests zu entwickeln, da wir auf das mühsame und sehr zeitaufwendige erstellen von den anderenfalls benötigten Testrümpfen verzichten können. Hierdurch ist es uns möglich deutlich schneller und effektiver zu testen.

7 Aufhebung und Wiederaufnahme

In diesem Abschnitt geht es darum, Kriterien zu definieren, die eine Unterbrechung aller oder eines bestimmten Tests dieses Testplans herbeiführen. Auch sind die Kriterien zu beachten, die für die Fortsetzung beziehungsweise die Wiederaufnahme der/des Tests erforderlich sind.

Die Test werden mit Junit unter NetBeans durchgeführt. Getestet wird sowohl unter Windows als auch unter Linux als Plattform. In dem Fall, dass es bei einem bestimmten Test zu Fehler wie zum Beispiel falsche Parameter, unerwartete Rückgaben, unvollständige Methoden oder andere Fehler kommt, führt das zu einer Unterbrechung des Tests. Der Test wird erst wieder fortgeführt, sobald Änderungen am entsprechenden Code durchgeführt wurden. Bei kritischen Fehlern, wie zum Beispiel ungeeignete Methoden, nicht realisierbare Methoden oder wenn die Tester feststellen, dass die getestete Methoden eine Abweichung von der in der Spezifikation festgelegten Softwaresystem haben, wird der Testphase in diesem Fall solange unterbrochen, bis Lösungen hierfür gefunden werden. Dieses beinhaltet eine komplette Neuimplementierung der betreffenden Abschnitte. Nach der Behebung der Abweichung wird das Testen fortgesetzt. Hierbei ist zunächst einmal darauf zu achten, dass es keine Abweichungen vom spezifizierten Verhalten gibt. Falls ein Testdurchlauf positiv ausfällt, es wird also ein Fehler entdeckt, muss zunächst einmal der Code überarbeitet werden. Der positive Test muss solange wiederholt werden bis kein Fehler mehr gefunden wird, der entsprechende Teil also fehlerfrei funktioniert.

8 Zu prüfendes Material

8.1 Hardware

An Hardware brauchen wir zunächst einmal mehrere Rechner mit ausreichend Rechenkapazität und Rechenleistung, die benötigte Hardware entspricht somit der in der Spezifikation bereits festgelegten Hardware. Für die Durchführung der Tests wird lediglich ein zusätzlicher Datenbankserver benötigt, um zu gewährleisten, dass die Software auch mit einem externen Datenbankserver lauffähig ist. Die Kommunikation zwischen den Komponenten wird dann durch eine TCP/IP Verbindung hergestellt, so dass entsprechende Netzwerkkabel benötigt werden.

8.2 Software

Um die Testphase durchführen zu können, brauchen wir zunächst einmal ein Betriebssystem und die entsprechende JVM. Um eine Portierbarkeit sicherzustellen, werden nicht alle Tests unter dem gleichen Betriebssystem durchgeführt werden. Es werden sowohl Tests unter einem windowsbasierenden Betriebssystem als auch unter einem linuxbasierenden Betriebssystem durchgeführt. Für die Testphase wird also an Software mindesten eine Windowsversion und eine Linuxversion benötigt. Hierfür nutzen wir sowohl den Rechner, den wir zu Hause haben, sollte in der Regel ein Rechner mit Windows Betriebssystem sein, als auch die an der Universität in Ebene 0 zur Verfügung gestellten Rechnern. Hierbei handelt es sich um Linux Rechner. Die Tests werden als auf verschiedenen Betriebssystemen durchgeführt. Weiterhin benötigen wir für die Testphase eine Entwicklungsumgebung. Hier kommt die gleiche Entwicklungsumgebung zum Einsatz, die auch für die Implementierung verwendet wird. Entgegen der bereits in der Spezifikation genannten Entwicklungsebene haben wir uns dazu entschlossen als Entwicklungsumgebung NetBeans zu verwenden. Wir haben uns hierzu entschlossen, da NetBeans eine wesentlich bessere Unterstützung von Tomcat bietet. Daher benötigen wir NetBeans in einer aktuellen Version. Zu Dokumentationszwecken benötigen wir eine Textverarbeitung. Um die Dokumentation sowohl unter Windows als auch unter Linux zu erleichtern und zu vereinheitlichen, haben wir uns bei der Textverarbeitung für Latex entschieden. Auch hiervon benötigen wir eine aktuelle Softwareversion.

8.3 Umgebung

Wie bereits bei der Software erwähnt, kommt bei uns die Entwicklungsumgebung NetBeans in der aktuellen Version zum Einsatz, da die Unterstützung von Tomcat

unter NetBeans deutlich einfacher ist.

8.4 Testwerkzeuge

Als Testwerkzeug wird wie vorgeschrieben JUnit zum Einsatz kommen. JUnit hilft bei der Fehlersuche und beim Erstellen der einzelnen Testfälle.

8.5 Andere Betriebsmittel

An weiteren Betriebsmitteln für die Durchführung der Testphase benötigen wir das notwendige Personal, um die Tests durchführen zu können. Weiterhin wird ein Arbeitsraum benötigt, in dem wir arbeiten können und der für Besprechungen genutzt werden kann. Hierfür stehen uns die Räume der Universität, insbesondere die 0. Ebene, so wie unsere eigene Wohnung zur Verfügung.

9 Testfälle

- *GUI* Die GUI (Grafische Benutzeroberfläche) Besteht im größten Teil aus JSP/HTML Seiten. Die Korrektheit der Darstellung wird vom Entwickler festgestellt. Die Namen der Interaktionselemente (Inputboxen, Radiobuttons etc.) wie auch der Klassen der Anwendungslogik sind für den ordentlichen Arbeitsablauf des Programms kritisch. Solche Elemente, die den regulären Ablauf einer Benutzersessions verhindern können, dürfen beim gegebenen UI nicht fehlen. Jeder Anwendungsfall muss in jeder Vorgangskette auf Funktionsfähigkeit geprüft werden. Dead-Links sind innerhalb diesen Pfaden nicht zuläßig.

- *UserManagement*

  addUser()
  Beim Einfgen eines Benutzers soll wie bei allen Einfüge-Operationen die Integrität des Eintrages geprüft werden. Es muss noch sichergestellt werden, dass sich der hiermit registrierte Benutzer sich beim System anmelden kann.

  editUser()
  Nach Änderung muss die erwartete Struktur erhalten sein. Ein Log-In muss danach (immer noch) möglich sein, es sei denn Benutzer wurde gesperrt.

  requestPassword()
  Es muss sichergestellt werden, dass das generierte Passwort das Email-Server-Programm erreicht. Ein Login-Vorgang mit dem versendeten Passwort muss erfolgreich abzuschließen sein.

  login()
  Der Bennutzer muss seinen Rechten entsprechenden Tätigkeiten ausführen können. Zugriff auf solche Funktionalität muss Unbekannten / Inkorrekten Nutzern verweigert werden.

  logout()
  Der Benutzer darf die vom Log-in bedingte Funktionen nicht zugreifen können

- *AdministrationModule*

  logIn()
  Sicher zu stellen ist, dass nach erfolgreichem Vorgang die Administratorfunktionen zugreifbar sind.

- *SearchEngine* Es muss gesichert werden, dass die Such-Methoden allen mï¿½lichen Treffer, einer bestimmten Anfrage entsprechend, zurückliefert.

- *ArticleManagement*

  addArticle()
  Diese Methode fügt einen neuen Artikel ein. Zu prüfen ist, ob nach erfolgreicher Ausführung (return=true) die erwartete Informationen in der Datenbank vorhanden sind. Falls der einzufügende *Article* invalide ist, darf die Methode nichts in der Datenbank eintragen.

  editArticle()
  Beim ersetzen von Informationen eines bestehenden Artikels dürfen die von ihm abhängigen Strukturen nicht beschädigt werden.

  importBibtex
  Diese Methode darf keinen invaliden BibTeX-String in der Datenbank aufnehmen. Falls der eingegebene Text korrekt ist, muss er an den entsprechenden Stellen in der Datenbank verteilt werden.

- *Taxonomy* Hierbei ist bei jeder Operation zu testen, ob die Abhängigkeiten unter den Kategorien erhalten werden. Hier gilt es auch, dass nur syntaktisch korrekte Eingaben zu Zustandsänderungen führen dürfen.

- *BibTexEntry*
  In dieser Struktur ist die Integrität der Listenkontainer (`requiredTags`, `optionalTags`) zu testen. Man muss auch noch testen, ob die Struktur `Importale` und `Exportable` ihrem Aufbau entsprechend implementiert ist. D.h. ist sie importier- und/oder exportierbar.

- *ImpEx*
  Bei diesem Modul muss getestet werden, ob die exportierte bzw. importierte Datei dem gewünschten Format entspricht. z.B.ensprechenden BibTeX, XML, Struktur, die in der Datenbank aufgenommen werden kann usw.

- *Article*

  erase(), edit()
  Bei diesen Methoden muss man prüfen, ob die am Löschen oder Einfügen teilnehmenden Strukturen ihre Integrität behalten. Die Funktionen dürfen z.B. die Taxonomiestruktur, die Artikelrelationen und weitere Abhängigkeiten nicht beschädigen.

  rate()
  Diese Methode muss ein validen Wert als Rating einsetzen.

10 Zeitplan

Wurde bereits abgegeben und ist von daher nicht noch mal hier aufgeführt.